Virtual memory for real-time systems using hPMP

Konrad Walluszik[1], Daniel Auge[1], Gerhard Wirrer[1], Holm Rauchfuss[1] and Thomas Röcker[1]

[1]Infineon Technologies AG, 85579 Neubiberg

Abstract

To satisfy automotive safety and security requirements, memory protection mechanisms are an essential component of automotive microcontrollers (MCU). In today's available systems, either a fully physical address-based protection is implemented utilizing a memory protection unit, or a memory management unit takes care of memory protection while also mapping virtual addresses to physical addresses. In this work, we showcase an extension to merge benefits from both systems to the current hPMP proposal, in order to address both requirements from Software Defined Vehicles (SDV) and realtime-applications.

Introduction

In automotive compute systems, the aspect of memory protection plays a distinctly important role for satisfying safety and security requirements. From a type-classification perspective, we distinguish mechanisms employing physical memory-only vs. virtual memory-based systems. The latter applying any kind of translation, i.e. effective addresses are 'mapped' to physical addresses based on a certain rule-set. Applying a selective set-based strategy, large physical memories can be addressed by virtual address ranges. In virtual memory-based systems, typically translation is performed page-based, i.e. blocks of predefined size (and located at certain physical addresses) get a virtual address assigned, and can consequently form contiguous (or non-contiguous) address maps though the individual blocks might physically scattered [1]. Such approach implies the need for a look-up mechanism, which is invoked upon every access; consequently, latency is induced to the system. In order to mitigate this effect, caching strategies can be applied which avoids potentially costly multi-stage lookups (also referred as table-lookups). While the caching increases performance of lookup *on average*, it becomes an additional burden when trying to analyze worst-case timings/boundaries due to the induced dependency on execution history.

Requirements for Realtime virtualization

In order to leverage the benefits of virtual memory for automotive applications with realtime-requirements, a solution is mandated which provides an address-translation feature minimizing additional complexity introduced to analysis of time-boundaries. (Note: The authors acknowledge that usage of virtualization of every manner adds complexity over purely physical solutions, yet at the benefit of reduced hardware cost.) Furthermore, the feature should be transparent to applications/setups for which virtual addressing is not required, i.e. change of the programming model of the standard memory protection unit (case of explicit protection) shall be avoided. Finally, impact to memory access timing needs to be avoided, especially when considering systems using fast local memories.

For our extension we assume the following model: H-extension w/o MMU ('Svbare') (see [2]) using hPMP (unified model) + vSPMP (see proposal in [3]). Address translation is only performed by hPMP, i.e. both guest- and user-code operate on guest physical addresses employing explicit protection for separation of VS and VU. Consequently, usage of the extension requires a modification to baseline-hypervisor, which manages the translation ruleset.

Configuration model

Figure 1 a) shows an exemplary address map (e.g. assumed for an embedded MCU) that includes regions for closely coupled memory (CCM) that hold instructions (I) and data (D). For storing program code and static data, a nonvolatile memory (NVM) region is considered. Furthermore, the MCU can include various different SRAM regions which can be utilized to store data during execution. A dedicated segment (e.g. placed at the top of the address space) is utilized for peripheral access. Figure 1 b) illustrates a minimalistic example of two VMs being managed by a hypervisor (HV). Considered are dedicated sections in DCCM used as stack, code regions in the NVM range and two global data regions scattered across available SRAM.

Based on the exemplary defined memory map, our target-configuration of hPMP is employing pairs of *pmpaddr*-registers, which have defined matching-mode OFF and TOR in *pmpcfg*, respectively. The *pmpaddr*-pairs define start- and end address of a protection region and permissions from *pmprange* mapping to TOR are considered. During reconfiguration, we consider

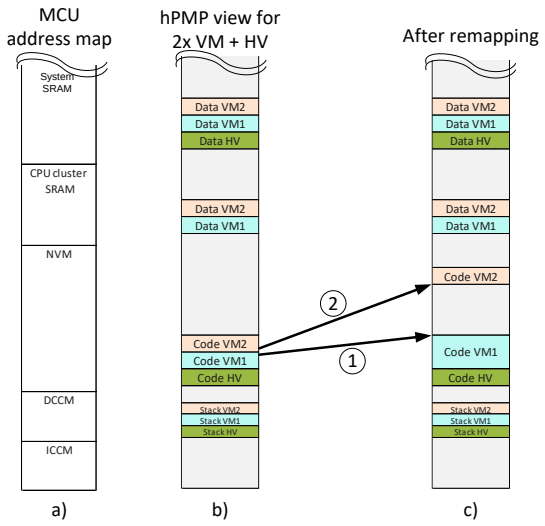

Figure 1: *a) Exemplary memory map of a microncontroller, b) Perspective of a hypervisor, c) Address redirection of hPMP*

respective ranges to be disabled by using the *pmpswitch*-register. We note that hPMP-implementations might be restricted to the OFF-TOR case (effectively only supporting A=0 and A=0,1 for even- and odd-numbered cfgs, respectively). We note that such scenario is due to requirements for dynamic reconfiguration and support of non-continuous address maps with no restriction over an implementation supporting all specified values for A.

The targeted permission model reads as follows: To all regions for the hypervsor VM-access is disallowed, protecting the HV from unintended modification and/or elevation of privilege. The VM-regions are configured with RW/RX for data and code-related regions, respectively. Considering the whitelisting-logic applied in the unified-model of sPMP for VS/VU, this requires rules with S=0 being used. Via *hpmpswitch* register, the HV will disable regions of all other VMs before scheduling next distinct VM (its regions are activated accordingly). The HV-execution itself is protected (e.g. to mitigate effects resulting from random hardware-faults) by dedicated ranges with RW/RX, yet using rules of type S=1 (activated in *spmpswitch* permanently).

Updating the hPMP configuration during VM switch can be more or less complex, depending on the number of implemented hPMP entries, the number of VMs running on the CPU and also the number and placement of memory regions to be separated for each VM. In cases where the number of needed hPMP entries for all regions to be separated is smaller or equal the number of implemented hPMP entries, the update can be performed using the hpmpswitch registers as indicated above.

hPMP extension

In this work, we introduce additional HV-CSRs named *hpmpoffsetx* (where x=0-63), encoding most significant bits in 34-bit address space of RV32. When executing in V=1, each *hpmpoffsetx* is used to derive virtual addresses from *hpmpaddrx*, considering a hit in a respective entry of hPMP. Contrary when V=0, the registers have no effect. In this sense, Guest-OS and -applications operate on guest-physical addresses, while HV is employing physical addressing.

In order to keep the mechanism lean, we assume hPMP-implementations to use of OFF-TOR strategy as described above. Then even-numbered offsets can be hardwired '0', while odd-numbered *hpmpoffsetx* applies to *hpmpaddrx* and *x-1*. Consequently, VMs can be 'moved' within virtual address space, while resizing requires change of respective *hpmpaddr*.

In the following example we show a scenario where the code section of VM1 increases its memory footprint from 512KB to 768KB (see arrow 1 in Figure 1). This can be the result of an update or feature extension for the application running on this virtual machine. As a consequence, it is required to move the image of VM2 to have nonoverlapping address regions (see arrow 2 in Figure 1). Instead of rebuilding/-linking the application of VM2, we employ the proposed concept of the hPMP to redirect all addresses of VM2 by a fixed offset.

Conclusion

This work highlights the limitations of classical virtual memory using a Memory Management Unit (MMU) approach within the context of real-time and deterministic systems. Through the demonstrated hpmp extension, we introduce an address redirection feature that enables the use of virtual memory while preserving the deterministic behavior of today's memory protection units. We currently focus on studying the behavior in corner cases (e.g. overlapping regions, other matching modes, etc.), feasibility (area, power, timing), and integration to hypervisor.

References

[1] John L. Hennessy and David A. Patterson, Computer Architecture - A Quantitative Approach, 6th edition, Morgan Kaufmann Publishers, 2019.

[2] Andrew W. et al, The RISC-V Instruction Set Manual. Volume II: Privileged Architecture v20211203, 2021.

[3] Sandro P. et al, "RISC-V Needs Secure "Wheels": the MCU Initiator-Side Perspective", https://arxiv.org/html/2410.09839v1, 2024.